```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

+ Code + Text

```
from sklearn.datasets import load_iris
```

```
load_iris()
```

```
         [7.4, 2.8, 6.1, 1.9],
         [7.9, 3.8, 6.4, 2. ],
         [6.4, 2.8, 5.6, 2.2],
         [6.3, 2.8, 5.1, 1.5],
         [6.1, 2.6, 5.6, 1.4],
         [7.7, 3. , 6.1, 2.3],
         [6.3, 3.4, 5.6, 2.4],
         [6.4, 3.1, 5.5, 1.8],
         [6. , 3. , 4.8, 1.8],
         [6.9, 3.1, 5.4, 2.1],
         [6.7, 3.1, 5.6, 2.4],
         [6.9, 3.1, 5.1, 2.3],
         [5.8, 2.7, 5.1, 1.9],
         [6.8, 3.2, 5.9, 2.3],
         [6.7, 3.3, 5.7, 2.5],
         [6.7, 3. , 5.2, 2.3],
         [6.3, 2.5, 5. , 1.9],
         [6.5, 3. , 5.2, 2. ],
         [6.2, 3.4, 5.4, 2.3],
         [5.9, 3. , 5.1, 1.8]]),
  'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
         2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
         2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
  'frame': None,
  'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
  'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Characteristics:**\n\n:Number of
 Instances: 150 (50 in each of three classes)\n:Number of Attributes: 4 numeric, predictive attributes and the class\n:Attribute
 Information:\n    - sepal length in cm\n    - sepal width in cm\n    - petal length in cm\n    - petal width in cm\n    -
 class:\n            - Iris-Setosa\n            - Iris-Versicolour\n            - Iris-Virginica\n\n:Summary
 Statistics:\n\n============== ==== ==== ======= ===== ====================\n                Min  Max   Mean    SD   Class
 Correlation\n============== ==== ==== ======= ===== ====================\nsepal length:   4.3  7.9   5.84   0.83
 0.7826\nsepal width:    2.0  4.4   3.05   0.43   -0.4194\npetal length:   1.0  6.9   3.76   1.76    0.9490  (high!)\npetal
 width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n============== ==== ==== ======= ===== ====================\n\n:Missing
 Attribute Values: None\n:Class Distribution: 33.3% for each of 3 classes.\n:Creator: R.A. Fisher\n:Donor: Michael Marshall
 (MARSHALL%PLU@io.arc.nasa.gov)\n:Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The dataset is
 taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning Repository, which has two
 wrong data points.\n\nThis is perhaps the best known database to be found in the\npattern recognition literature.  Fisher\'s
 paper is a classic in the field and\nis referenced frequently to this day.  (See Duda & Hart, for example.)  The\ndata set
 contains 3 classes of 50 instances each, where each class refers to a\ntype of iris plant.  One class is linearly separable from
 the other 2; the\nlatter are NOT linearly separable from each other.\n\n.. dropdown:: References\n\n  - Fisher, R.A. "The use of
 multiple measurements in taxonomic problems"\n    Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n
 Mathematical Statistics" (John Wiley, NY, 1950).\n  - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene
 Analysis.\n    (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.\n  - Dasarathy, B.V. (1980) "Nosing Around the
 Neighborhood: A New System\n    Structure and Classification Rule for Recognition in Partially Exposed\n    Environments".  IEEE
 Transactions on Pattern Analysis and Machine\n    Intelligence, Vol. PAMI-2, No. 1, 67-71.\n  - Gates, G.W. (1972) "The Reduced
 Nearest Neighbor Rule".  IEEE Transactions\n    on Information Theory, May 1972, 431-433.\n  - See also: 1988 MLC Proceedings,
 54-64.  Cheeseman et al"s AUTOCLASS II\n    conceptual clustering system finds 3 classes in the data.\n  - Many, many more
 ...\n',
  'feature_names': ['sepal length (cm)',
   'sepal width (cm)',
   'petal length (cm)',
   'petal width (cm)'],
  'filename': 'iris.csv',
  'data_module': 'sklearn.datasets.data'}
```

```
df=load_iris()
```

```
datasets=pd.DataFrame(df.data)
print(datasets)
```

```
       0    1    2    3
0    5.1  3.5  1.4  0.2
1    4.9  3.0  1.4  0.2
2    4.7  3.2  1.3  0.2
3    4.6  3.1  1.5  0.2
4    5.0  3.6  1.4  0.2
..   ...  ...  ...  ...
```

```
145  6.7  3.0  5.2  2.3
146  6.3  2.5  5.0  1.9
147  6.5  3.0  5.2  2.0
148  6.2  3.4  5.4  2.3
149  5.9  3.0  5.1  1.8

[150 rows x 4 columns]
```

```
datasets.columns=df.feature_names
```

```
datasets.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Next steps: Generate code with datasets | View recommended plots | New interactive sheet

```
## here i will keep my independent features X and dependent features as y
X=datasets
y=df.target
```

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
## train test split we do here

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

```
X_train
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
81	5.5	2.4	3.7	1.0
133	6.3	2.8	5.1	1.5
137	6.4	3.1	5.5	1.8
75	6.6	3.0	4.4	1.4
109	7.2	3.6	6.1	2.5
...	...	...	...	...
71	6.1	2.8	4.0	1.3
106	4.9	2.5	4.5	1.7
14	5.8	4.0	1.2	0.2
92	5.8	2.6	4.0	1.2
102	7.1	3.0	5.9	2.1

105 rows × 4 columns

Next steps: Generate code with X_train | View recommended plots | New interactive sheet

```
# standardizing the dataset by this output will be correct
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

```
X_train=scaler.fit_transform(X_train)
```

```
X_test=scaler.transform(X_test)
```

```
X_train

## scaler.inverse_transform(X_train) we can get back from standardizing
```

```
       [-0.29283662, -0.74242333,  0.19235097,  0.08245999],
       [-0.89573553,  0.93659559, -1.38370397, -1.40568508],
       [-0.17225683, -0.02284379,  0.19235097, -0.05282593],
       [ 2.23933883,  1.89603497,  1.65166111,  1.30003323],
       [-1.49863445,  0.4568759 , -1.44207638, -1.40568508],
       [ 0.43064208, -0.26270364,  0.25072338,  0.08245999],
       [-0.17225683, -1.22214302,  0.65933022,  1.0294614 ],
       [-0.4134164 ,  2.85547435, -1.44207638, -1.40568508],
       [ 0.18948252, -0.02284379,  0.54258541,  0.75888956],
       [-0.05167705, -0.74242333,  0.71770262,  0.89417548],
       [ 0.18948252, -1.94172256,  0.07560616, -0.32339776],
       [-0.53399618, -0.02284379,  0.36746819,  0.35303182],
       [ 0.43064208,  0.93659559,  0.89281984,  1.43531914],
       [-0.4134164 , -1.70186271,  0.07560616,  0.08245999],
       [-0.53399618,  2.13589481, -1.26695916, -1.13511325],
       [-1.01631531, -1.70186271, -0.33300068, -0.32339776],
       [ 0.67180165, -0.74242333,  0.83444743,  0.89417548],
       [-1.01631531,  0.69673574, -1.44207638, -1.40568508],
       [-1.01631531,  0.4568759 , -1.55882119, -1.40568508],
       [-0.4134164 , -1.46200287, -0.04113865, -0.18811184],
       [ 1.033541  , -0.02284379,  0.65933022,  0.62360365],
       [-1.1368951 ,  0.21701605, -1.38370397, -1.40568508],
       [-0.05167705, -0.50256349,  0.71770262,  1.57060506],
       [-1.01631531,  0.93659559, -1.38370397, -1.40568508],
       [-1.01631531,  1.17645543, -1.32533157, -0.86454142],
       [ 0.06890273,  0.4568759 ,  0.54258541,  0.75888956],
       [-0.89573553, -1.22214302, -0.50811789, -0.18811184],
       [ 1.27470056,  0.4568759 ,  1.06793706,  1.43531914],
       [ 0.18948252, -0.74242333,  0.71770262,  0.48831773],
       [ 0.3100623 , -0.98228318,  1.00956465,  0.2177459 ],
       [ 2.23933883, -0.02284379,  1.30142668,  1.43531914],
       [-0.4134164 , -1.22214302,  0.07560616,  0.08245999],
       [-1.73979401, -0.26270364, -1.44207638, -1.40568508],
       [-1.8603738 , -0.02284379, -1.6171936 , -1.540971  ],
       [ 0.18948252, -1.94172256,  0.65933022,  0.35303182],
       [ 1.63643991,  0.4568759 ,  1.24305427,  0.75888956],
       [-1.49863445,  0.21701605, -1.38370397, -1.40568508],
       [-0.89573553,  1.17645543, -1.44207638, -1.27039917],
       [-1.73979401, -0.02284379, -1.50044878, -1.40568508],
       [ 0.55122187, -1.22214302,  0.60095781,  0.35303182],
       [ 0.55122187,  0.93659559,  1.00956465,  1.57060506],
       [-1.49863445,  0.93659559, -1.44207638, -1.27039917],
       [ 1.15412078, -0.02284379,  0.95119225,  1.16474731],
       [ 0.55122187,  0.69673574,  1.24305427,  1.70589097],
       [-1.37805466,  0.4568759 , -1.50044878, -1.40568508],
       [ 0.3100623 , -0.26270364,  0.484213  ,  0.2177459 ],
       [ 0.79238143, -0.50256349,  0.4258406 ,  0.35303182],
       [ 0.43064208, -0.50256349,  0.54258541,  0.75888956],
       [ 1.39528035,  0.4568759 ,  0.484213  ,  0.2177459 ],
       [ 0.67180165,  0.4568759 ,  0.83444743,  1.43531914],
       [-0.89573553,  1.89603497, -1.32533157, -1.40568508],
       [ 1.27470056,  0.21701605,  0.89281984,  1.16474731],
       [ 0.06890273, -0.02284379,  0.19235097,  0.35303182],
       [ 0.79238143, -0.02284379,  0.77607503,  1.0294614 ],
       [-0.17225683, -0.98228318, -0.21625586, -0.32339776],
       [-0.77515575, -0.74242333,  0.01723376,  0.2177459 ],
       [ 0.3100623 , -0.02284379,  0.4258406 ,  0.2177459 ],
       [-1.61921423, -1.70186271, -1.50044878, -1.27039917],
```

```
## we implement here the linear regression
## we import the linear regression from sklearn
from sklearn.linear_model import LinearRegression
# cross validation
from sklearn.model_selection import cross_val_score
```

```
regression=LinearRegression()
regression.fit(X_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
mse=cross_val_score(regression,X_train,y_train,scoring='neg_mean_squared_error',cv=5)
```

```
np.mean(mse)
```

```
np.float64(-0.05617171940989511)
```

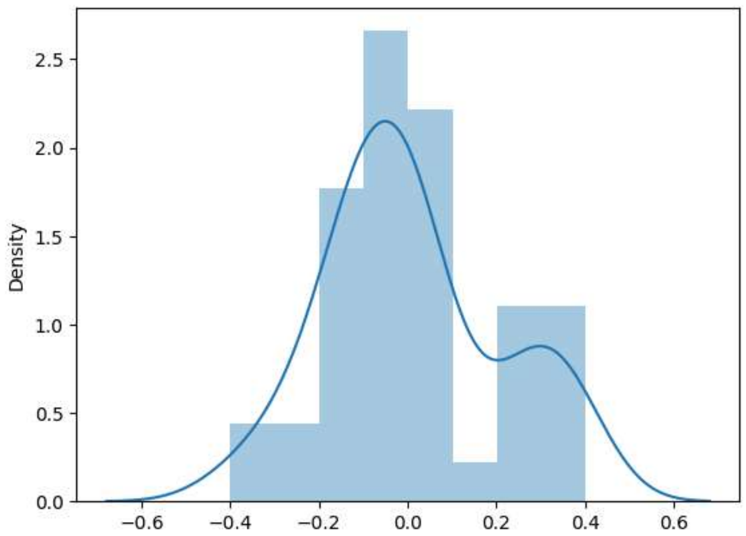
```
#prediction

reg_predict=regression.predict(X_test)
```

```
reg_predict
```

```
array([ 1.24069097, -0.04537609,  2.24501083,  1.35143666,  1.29775083,
        0.01024241,  1.05031173,  1.82525399,  1.37084413,  1.06699186,
        1.70363485, -0.08712067, -0.165166  , -0.07724353, -0.03380619,
        1.40167699,  2.00651252,  1.04725931,  1.28368327,  1.97600474,
        0.01782354,  1.59952875,  0.079732  ,  1.92307532,  1.8621986 ,
        1.8790815 ,  1.80251247,  2.04196713,  0.01873817,  0.01291496,
       -0.15365607, -0.08046738,  1.18506728, -0.00461982, -0.02934265,
        1.68665136,  1.29088786, -0.07995434, -0.09076782, -0.16795331,
        1.75520461,  1.37514144,  1.3174234 , -0.07193336, -0.1131512 ])
```

```
import seaborn as sns
sns.distplot(reg_predict-y_test)
plt.show()
```

```
<ipython-input-22-0c78bb25d2d9>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(reg_predict-y_test)
```




```
from sklearn.metrics import r2_score
score=r2_score(reg_predict,y_test)
```

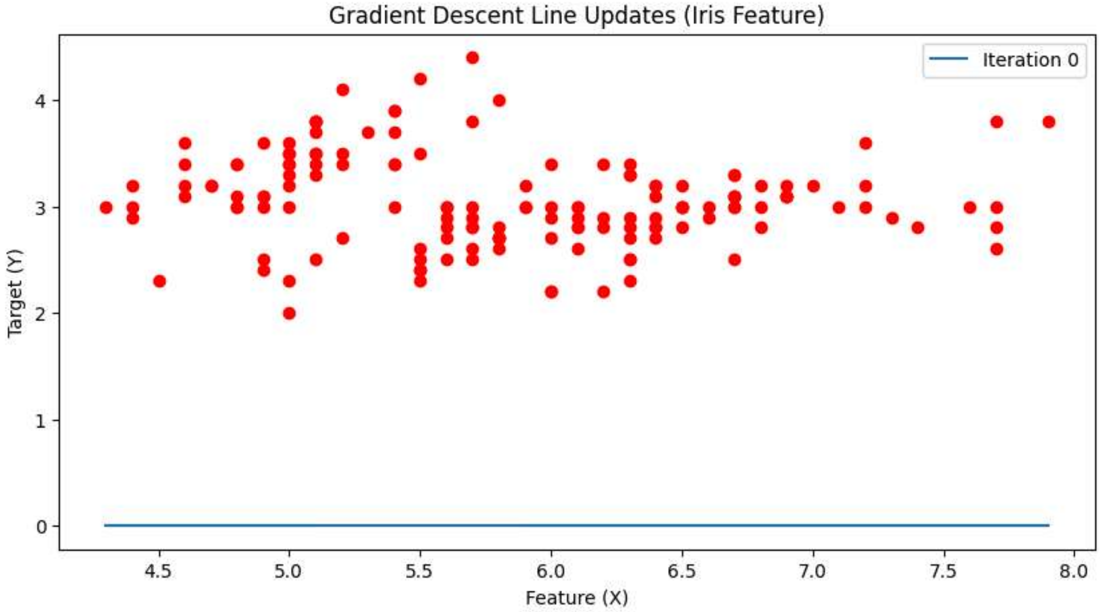
```
score
```

```
0.9453000057840795
```

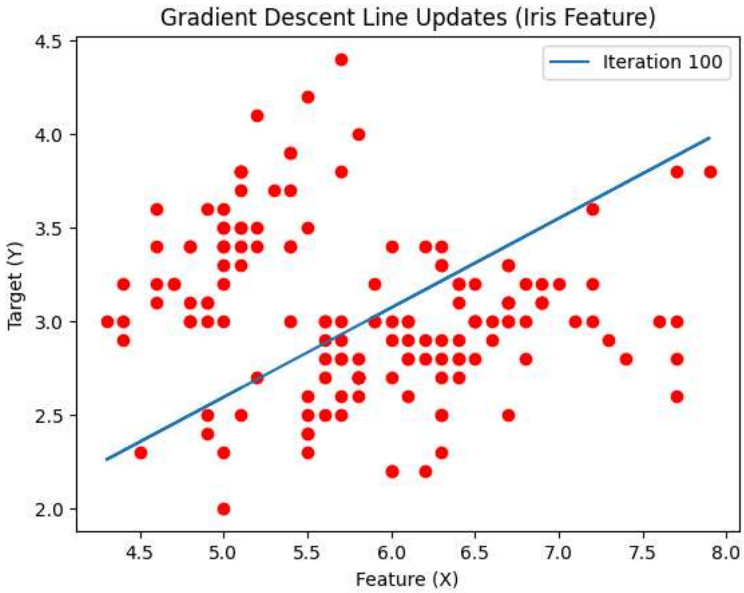
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

def gradient_descent_visualization(x, y, learning_rate=0.01, iterations=1000):
    m_curr = 0
    b_curr = 0
    n = len(x)
    cost_history = []

    plt.figure(figsize=(10, 5))

    for i in range(iterations):
        y_predicted = m_curr * x + b_curr
        cost = (1/n) * sum((y - y_predicted) ** 2)
        cost_history.append(cost)
```

```
        md = (-2/n) * sum(x * (y - y_predicted))
        bd = (-2/n) * sum(y - y_predicted)

        m_curr = m_curr - learning_rate * md
        b_curr = b_curr - learning_rate * bd

        # Plot every 100 iterations
        if i % 100 == 0:
            print(f"Iteration {i} | Cost: {cost}")
            plt.scatter(x, y, color="red")
            plt.plot(x, y_predicted, label=f"Iteration {i}")
            plt.xlabel("Feature (X)")
            plt.ylabel("Target (Y)")
            plt.title("Gradient Descent Line Updates (Iris Feature)")
            plt.legend()
            plt.pause(0.5)
            plt.clf()

    # Final best fit line
    y_predicted = m_curr * x + b_curr
    plt.scatter(x, y, color="red")
    plt.plot(x, y_predicted, color="green", label="Best Fit Line")
    plt.xlabel("Feature (X)")
    plt.ylabel("Target (Y)")
    plt.title("Final Best Fit Line on Iris Data")
    plt.legend()
    plt.show()

    # Plot cost function convergence
    plt.figure(figsize=(8, 5))
    plt.plot(range(iterations), cost_history, color='blue')
    plt.xlabel("Iterations")
    plt.ylabel("Cost")
    plt.title("Cost Function Convergence")
    plt.show()

    print(f"Final slope (m): {m_curr}, intercept (b): {b_curr}")

# Load Iris dataset
iris = load_iris()
X = iris.data[:, 0]   # sepal length (feature)
Y = iris.data[:, 1]   # sepal width (target)

gradient_descent_visualization(X, Y)
```

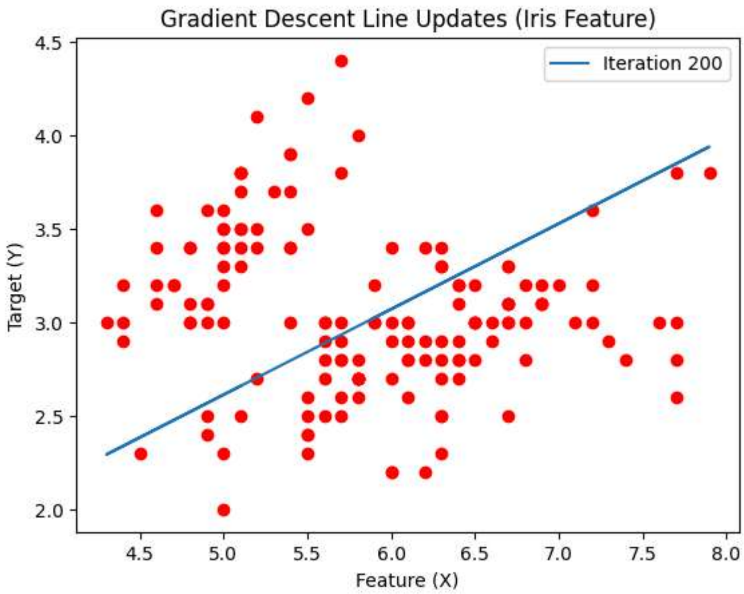
Iteration 0 | Cost: 9.535999999999994



Iteration 100 | Cost: 0.3877364149031411



Iteration 200 | Cost: 0.37296089599581966



Iteration 300 | Cost: 0.35926812147799686